TASK 1

Approximate Nearest Neighbour with Locality Sensitive Hashing

You are in the group Randomorder consisting of alobbe (alobbe@student.ethz.ch (mailto://[u'alobbe@student.ethz.ch'])), deliaf (deliaf@student.ethz.ch (mailto://[u'deliaf@student.ethz.ch'])) and sridharg (sridharg@student.ethz.ch (mailto://[u'sridharg@student.ethz.ch'])).

1. READ THE TASK DESCRIPTION

2. SUBMIT SOLUTIONS

3. HAND IN FINAL SOLUTION

1. TASK DESCRIPTION

In this project, we are interested in detecting near-duplicate html pages. Suppose you want to write your own search engine. For reliability reasons, many web pages store duplicates or near duplicate versions of the same information. Your task is to develop an efficient method to to detect whether a pages are near-duplicates of each other so that your search engine does not present the user redundant search results.

INPUT AND OUTPUT SPECIFICATION

Download handout (/static/task1.zip)

We will use Jaccard similarity based on the page features as the similarity metric. You are given two text files:

1. **handout_shingles.txt**: Each line contains the features for one page file and is formatted as follows: **PAGE_XXXXXXXXX** followed by a list of space delimited integers in range **[0, 8192]**. You can consider them equivalent to shingles in the context of near-duplicate document retrieval.
2. **handout_duplicates.txt**: Each line contains a pair of near duplicates (pages that are at least 85% similar according to the Jaccard similarity). Each line is a tab-delimited pair of integers **where the first integer is always smaller**. This file is used to measure the error of your output as described below.

Your goal is to develop a Locality Sensitive Hashing program in the *MapReduce* setting. To facilitate development, we provide you with a (small) *MapReduce* implementation: The **runner.py** script allows you to run a *MapReduce* program and measure the performance of the produced solutions.

To create a valid *MapReduce* program for this task, you need to create a Python source file that contains both a mapper and a reducer function. The **mapper(key, value)** function takes as input a (key, value) tuple where key is None and value is a string. It should yield (key, value) pairs. The **reducer(key, value)** function takes as input a key and a list of values. It should yield (key, value) pairs. A skeleton of such a function is provided in the **example.py**.

You are free to implement the mapper and reducer in any way you see fit, as long as the following holds:

1. The maximum number of hash functions per mapper used is 1024.
2. The cummulative runtime of both mappers and reducers is limited to 3 minutes.
3. Each mapper receives a key and value pair where key is always None (for consistency). Each value is one line of input as described above.
4. Reducer should output a key, value pair of two integers representing the ID's of duplicated pages, the smaller ID being the key and the larger the value.
5. You may use the Python 2.7 standard library and **NumPy**. You are not allowed to use multithreading, multiprocessing, networking, files and sockets.

EVALUATION AND GRADING

For each line of the output of your algorithm we check whether the reported pages are in fact at least 85% similar. If they are, this output will count as one true positive. If they are not, it will count as one false positive. In addition, each pair of approximate neighbors that was not reported by your algorithm will count as one false negative. Given the number of true positives, false positives and false negatives, TP, FP and FN respectively, we can calculate:

1. **Precision**, also referred to as Positive predictive value (PPV), as **P = TP/(TP+FP).**
2. **Recall**, also referred to as the True Positive Rate or Sensitivity, as **R = TP/(TP+FN).**

Given precision and recall we will calculate the F1 score defined as **F1 = 2PR/(P + R).** We will compare the F1 score of your submission to two baseline solutions: A weak one (called **baseline easy**) and a strong one (**baseline hard**). These will have the F1 score of FBE and FBH respectively, calculated as described above. Both baselines will appear in the rankings together with the F1 score of your solutions.

Your grade on this task depends on the solution and the description that you hand in. As a rough (non-binding) guidance, if you hand in a properly-written description and your handed-in submission performs better than the easy baseline, you will obtain a grade exceeding a 4. If in addition, your submission beats the hard baseline, you obtain a 6.